Unica Plan V12.1.3 Integration Module

Contents

Chapter 1. What is Unica Plan Integration Services?

Unica Plan Integration Services combines the Unica Plan Integration Webservice, SOAP API procedures, and triggers to extend business capabilities.

Unica Plan Integration Services is a composite of the following.

- **Unica Plan Integration Webservice**

  Integration Services provide a way for Unica Plan customers and Professional Services to integrate Unica Plan with other applications that run in their environment.
- **Unica Plan procedures and SOAP API**

  Custom procedures can be defined within Unica Plan to extend Unica Plan business logic in arbitrary ways. After you define procedures, these procedures can be the targets for the Integration Services webservice calls from other applications. Procedures also can be defined to send messages to other applications.
- **Unica Plan triggers**

  Triggers can be associated with events and procedures in Unica Plan. When one such event occurs, the associated trigger is run.

REST APIs do not use Unica Plan integration services. For information about the REST API, see the Unica Plan Administrator's Guide.

Versions and backwards-compatibility

Future versions of the integration services will be backwardly compatible with all minor and maintenance releases that share a major version number. However, reserves the right to break compatibility with an earlier version for dot zero (x.0) major releases if the business or technical case warrants.

The major version number of this API is incremented if any of the following changes are made.

- Data interpretation changes
- Business logic changes (for example, service method functions changes)
- Method parameters, return types, or both change

The minor version number of the API is incremented if any of the following changes are made. These changes are compatible with an earlier version by definition.

- New method added
- New data type is added and its usage is restricted to a new method
- New element added to an enumerated type
- A new version of an interface is defined with a version suffix

Authentication

Authentication is not required; all clients are associated with a known Unica Plan user named PlanAPIUser. A system administrator configures the security capabilities of this special user to meet the needs of all webservice clients.

Locale

The only locale that is supported is the locale that is currently configured for the Unica Plan system instance. All locale-dependent data, such as messages and currency, are assumed to be in the system locale.

State management

The API and webservice are stateless; no per-client information is saved by the service implementation across API calls. This feature provides for an efficient service implementation and simplifies cluster support.

Database transactions

Unica Plan Integration Services does not show database transactions to the client, but uses such information if it is included in the execution context. If a transaction is started, then the effect of all API calls within a particular procedure is atomic. In other words, a failed API call leaves the database in the same state as if the API was never called at all. Other

users of Unica Plan do not see the changes until the procedure successfully completes the transaction.

API calls that update the database must first acquire an edit lock to prevent other users from modifying the underlying data during the API calls. Other users cannot update locked components until the API call completes. Likewise, the next Unica Plan user or API client must acquire the lock on the data before another API call is submitted.

Event processing

Operations on Unica Plan components through the API generate the same events as if a Unica Plan user did the operation. Users that subscribed to certain notifications, such as project state changes, are notified of state changes that result from API calls and user actions.

What are the requirements for Unica Plan Integration Services?

Unica Plan Integration Services has the following requirements.

Unica Plan Integration Services must:

- Loosely couple system integration.
- Provide a mechanism for customer applications to affect Unica Plan through webservice calls.
- Provide a mechanism for customer applications to be notified of certain events in Unica Plan.
- Provide a simple programming model that is easy to understand and use.
- Be robust when recovering from failure.
- Guarantee data integrity.
- Integrate with, and minimize the effect on, existing Unica Plan GUI-based customers.
- Provide fine-grained access to Unica Plan components while insulating programmers from underlying implementation details.
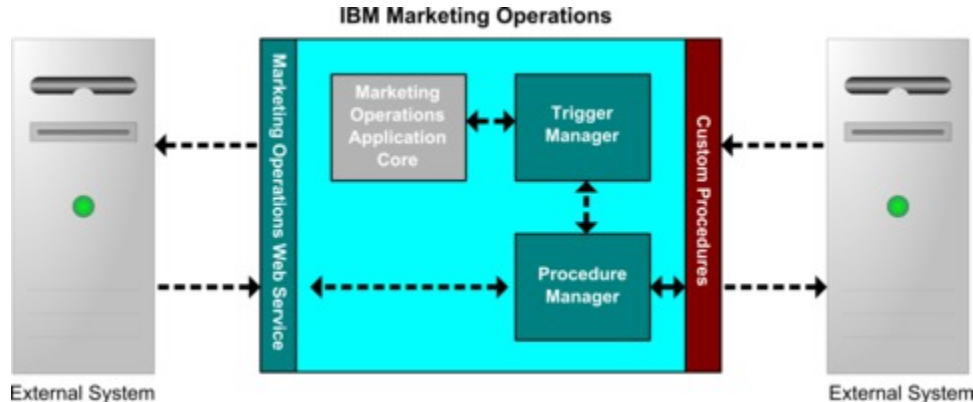
Unica Plan Integration Services basics

You use Unica Plan Integration Services to create custom procedures. You can use these procedures to trigger external events when certain events occur within Unica Plan. You can use these procedures to run Unica Plan functions from external systems or programs.

The API interface interacts with Unica Plan at the programmatic level, in the same way the GUI interfaces with Unica Plan at a user level. Using the API, you construct procedures. Using these procedures, you communicate between Unica Plan and external systems. The Unica Plan Webservice is the container object for the procedures, API, and triggers.

The architecture of the Unica Plan Integration Services is shown here.

The following are key components of the Integration Services.

- Unica Plan Procedure Manager: extends the business logic by interacting with Unica Plan through the API.
- Unica Plan Trigger Manager: associates a condition (for example, the state change of a marketing object) with an action (a procedure to run when the condition for the trigger is met).

Methods

You use the components of Unica Plan Integration Services to develop custom procedures, as shown in the following diagram.
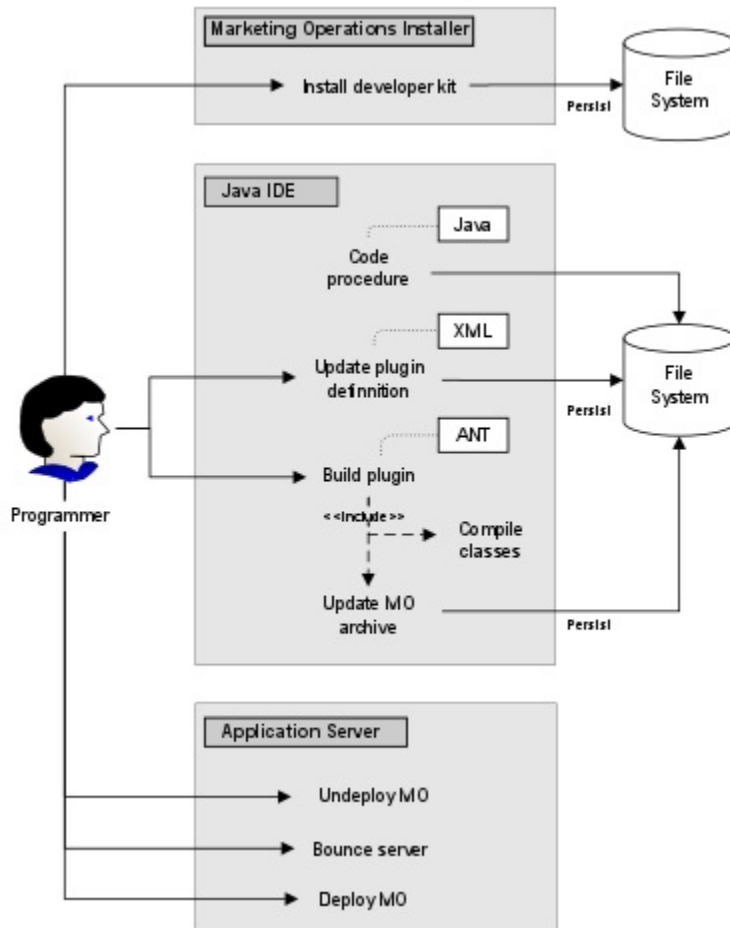
After you install the developer's kit, you follow these basic steps:

1. Code the custom procedure.
2. Update the plug-in definition in the XML definition file.
3. Build the plug-in:
    a. Compile the necessary classes.
    b. If you are using a third-party library that is not in the Unica Plan archive, bundle the library inside the `plan.war` file and redeploy.
4. Restart Unica Plan. Changes to the procedure classes are applied when you restart the application server.

> ✏️ **Note:** If you change the **plan.war** file, you must undeploy and redeploy Unica Plan with the new **plan.war** file. Undeploy and redeploy Unica Plan if you use a third-party library that is not in the Unica Plan archive and you edit the **plan.war** file.

Basic Example to communicate between Unica Plan and the API

The following basic example describes establishing communication between the API and Unica Plan. It does not do any useful work; it performs a round trip between Unica Plan and the Integration Services.

This example uses portions of the example procedures included with the Unica Plan Integration Services developer's kit. Specifically, you can find the code that is referenced here in the following files.

- PlanClientFacade.java
- PlanWSNOOPTestCase.java

The noop method is a webservice call to Unica Plan. It is defined in the PlanClientFacade class, and passes null values in an array.

```
public ProcedureResponse noop(String jobId)
  throws RemoteException, ServiceException {
  NameValueArrays parameters =
    new NameValueArrays(null, null, null, null, null, null, null, null);
  return _serviceBinding.executeProcedure("uapNOOPProcedure", jobId,
 parameters);
}
```

The procedure testExecuteProcedure calls the noop method from PlanClientFacade to establish a round trip with the Unica Plan application.

```
public void testExecuteProcedure() throws Exception {
  // Time out after a minute
```

```
 int timeout = 60000;

 PlanClientFacade clientFacade = new PlanClientFacade(urlWebService,

timeout);

 System.out.println("noop w/no parameters");

 long startTime = new Date().getTime();

 ProcedureResponse response = clientFacade.noop("junit-jobid");

 long duration = new Date().getTime() - startTime;


 // zero or positive status => success

 System.out.println("Status: " + response.getStatus());

 System.out.println("Duration: " + duration + " ms");

 assertTrue(response.getStatus() >= 0);

 System.out.println("Done.");

}
```

For details of NameValueArrays, ProcedureResponse, and other listed methods and data types, refer to the Unica Plan Integration Module and the JavaDocs.

Installing Integration Services

The Unica Plan Integration Services module is a separate, paid component. If you purchase the Integration Services module, you must install it.

1. Download the Unica Plan Integration Services installers.
2. The Unica installers detect the Integration Services module.
3. The installer sets configuration properties at **Plan** | **umoConfiguration** | **integrationServices** | **enableIntegrationServices**. You can customize your installation by changing configuration parameters. For more information, see Configuration parameters *(on page 22)*.

Software developer kit contents

The software developer kit contains documentation containing all publicapi classes and interfaces, and example code.

For the SOAP API, all the Unica Plan Integration Services components are installed under a folder labeled `devkits`.

Example code is installed in the following folders.

- The **build** folder contains scripts to build and deploy custom procedures.
- The **Classes** folder contains the compiled procedure classes.

  Users must deploy the compiled classes of their custom procedures at the path that is specified by the configuration parameter **integrationProcedureClasspathURL**. Then, the Unica Plan Procedure Manager loads them as specified in the `procedure-plugins.xml` configuration file.
- The **lib** folder contains the necessary libraries for developing and compiling custom procedures.
- The **src** folder contains source files for custom procedures. Users can place custom procedures to be started as triggers or web-services here. Only the SOAP API supports custome procedures.
  - The **src/procedure** folder contains `procedure-plugins.xml` configuration file. Every custom procedure that runs as a trigger based an event or through an external web-service must have an entry in this file. The entries must contain a fully qualified class path of procedure and required initialization parameters.
  - The **src/procedure** folder also contains some sample procedures that are included with Unica Plan. These procedures can be used to understand and develop your custom procedures.

    Place custom procedures under the **src** directory in a new folder structure, such as `com/<mycompany>/<mypackage>`. Do not place custom procedures in the sample procedures folder.
  - The **src/soap** folder contains sample web service clients that are developed in Java. Use these samples as a starting point for developing web service-based clients for Integration Services. This folder also contains binary scripts to start sample clients over the command line.

Hosted JavaDocs

For specific information about the public API methods, refer to the iPlanAPI class in the JavaDocs API documentation files.

These files are available in the following ways:

- By the files in the `<HCL_Unica>/<Plan_Home>/devkits/integration/javadocs` directory for the SOAP API on the server that hosts Unica Plan.
- By logging in to Unica Plan and selecting **Help > Product Documentation** from any page, and then downloading the `<version>PublicAPI.zip` file for the SOAP API.

Unica Plan documentation and help

Different people in your organization use Unica Plan to accomplish different tasks. Information about Unica Plan is available in a set of guides, each of which is intended for use by team members with specific objectives and skill sets.

The following table describes the information available in each guide.

Table 1. Guides in the Unica Plan documentation set

The following three-column table describes tasks in one column, guide names in the second column, and audience in the third column.

If you	See	Audience
• Plan and manage projects • Establish workflow tasks, milestones, and personnel • Track project expenses • Get reviews and approvals for content	Unica Plan User's Guide	• Project managers • Creative designers • Direct mail marketing managers • Marketers

Table 1. Guides in the Unica Plan documentation set

The following three-column table describes tasks in one column, guide names in the second column, and audience in the third column.

(continued)

If you	See	Audience
• Produce reports • Create to-dos and checklists		
• Design templates, forms, attributes, and metrics • Customize the user interface • Define user access levels and security • Implement optional features • Configure and tune Unica Plan	Unica Plan Administrator's Guide	• Project managers • IT administrators • Implementation consultants
• Create marketing campaigns • Plan offers • Implement integration between Unica Plan and Unica Campaign • Implement integration between Unica Plan and IBM Digital Recommendations	Unica Plan and Integration Guide	• Project managers • Marketing execution specialists • Direct marketing managers

Table 1. Guides in the Unica Plan documentation set

The following three-column table describes tasks in one column, guide names in the second column, and audience in the third column.

(continued)

If you	See	Audience
• Learn about new system features • Research known issues and workarounds	Unica Plan Release Notes®	Everyone who uses Unica Plan
• Install Unica Plan • Configure Unica Plan • Upgrade to a new version of Unica Plan	Unica Plan Installation Guide	• Software implementation consultants • IT administrators • Database administrators
Create custom procedures to integrate Unica Plan with other applications	Unica Plan Integration Module and the API JavaDocs available when you click **Help > Product Documentation** in Unica Plan, and then download the `UnicaPlan<version>PublicAPI.zip` file for the SOAP API and `UnicaPlan<version>PublicAPI-RestClient.zip` for the REST API.	• IT administrators • Database administrators • Implementation consultants
Learn about the structure of the Unica Plan database	Unica Plan System Schema	Database administrators
Need more information while you work	• Get help and search or browse the *User's, Administrator's*, or *Instal-*	Everyone who uses Unica Plan

Table 1. Guides in the Unica Plan documentation set

The following three-column table describes tasks in one column, guide names in the second column, and audience in the third column.

(continued)

If you	See	Audience
	lation guides: Click **Help > Help for this page** • Access all of the Unica Plan guides: Click **Help > Product Documentation** • Access guides for all Unica products: Click **Help > All Unica Suite Documentation**	

Chapter 2. Unica Plan Integration Webservice

The webservice provides a client view of the Unica Plan Integration Services, which is part of the deployment of the Unica Plan server. The service is used concurrently with Unica Plan web users.

The webservice supports one API call, executeProcedure.

A client makes this webservice call directly.

Unica Plan Integration Services WSDL

The Web Services Definition Language (WSDL) was defined by hand and is the final word on the webservice definition.

Axis

This version of the webservice uses Axis2 1.5.2 to generate the server-side classes that make up the web service implementation from the WSDL file. Users can use any version of Axis, or a non-Axis technique, to create a client side implementation for integrating with the API from the supplied WSDL.

Protocol version

The version of the protocol is explicitly bound to the WSDL as follows:

- As part of the WSDL name, for example, `PlanIntegrationService1.0.wsdl`
- As part of the WSDL targetNamespace, for example, `xmlns:tns="http://webservices.unica.com /MktOps/services/PlanIntegrationServices1.0?wsdl"`

WSDL

One WSDL file is provided with Unica Plan Integration Services: `PlanIntegrationServices1.0.wsdl`. The WSDL is delivered in the `integration/examples/soap/plan` directory. The example build script uses this file to generate the appropriate client-side stubs to connect to the webservice.

executeProcedure

executeProcedure is the on API call that is supported by the webservice.

Syntax

```
executeProcedure(string key, string jobid, NameValueArrays paramArray)
```

Returns

```
int: status

Message[]: messages
```

Description

This method invokes the specified procedure with an optional array of parameters. The call executes synchronously; that is, it blocks the client and returns the result upon completion.

Parameters

Table 2. executeProcedure parameters

Name	Description
key	The unique key of the procedure to run. A *RemoteException* error is returned if no procedure is bound to **key**.
jobid	Optional string that identifies the job that is associated with this procedure execution. This string is a pass-through item, but it can be used to tie client jobs to the execution of a particular procedure.
paramArray	An array of parameters to pass to the procedure. An error status and message is returned if one or more of the parameters is invalid (such as, the wrong type or an incorrect value). It is up to the client to determine the parameters, their types, and the number of values that are required by the procedure.

Return Parameters

Table 3. executeProcedure return parameters

Name	Description
status	An integer code: • 0 indicates that the procedure ran successfully • an integer indicates an error Procedures can use the status to indicate different levels of errors.
messages	An array of zero or more message data structures. If **status** is 0, this array does not contain ERROR messages, but might contain INFOR-MATION and WARNING messages. If **status** is non-zero, messages can contain any mix of ERROR, IN-FORMATION, and WARNING messages.

Unica Plan Integration Webservice data types

The data types that are used by the webservice are independent of any particular service binding or programming implementation.

The following notation is used.

- *<type>*: *<type definition>* defines a simple data type. For example:

  Handle: string
- *<type>*: [*<type definition>*] defines a complex data type or a data structure.
- *<type>*: { *<type definition>* } defines a complex data type or a data structure.

Complex type elements and API parameters can use these types to declare arrays. For example:

```
Handle [] handles
```

The type, handles, is an array of Handle types.

Primitive types

Primitive types are restricted to the types defined in the table that follows to simplify support for SOAP 1.1 bindings. All types can be declared as arrays, for example, **String [ ]**. Inherently, binary data types, such as **long**, can be represented as strings by a protocol binding (for example, SOAP). This representation, however, has no effect on the semantics of the type, permissible values, and so on, as seen by the client.

Table 4. Primitive types

API Type	Description	SOAP Type	Java™ Type
Boolean	Boolean value: **true** or **false**	xsd:Boolean	Boolean
dateTime	A date time value	xsd:datetime	Date
decimal	An arbitrary-precision, decimal value	xsd:decimal	java.math.BigDeci- mal
double	A double-precision, signed, decimal value	xsd:double	double
int	A signed, 32-bit, integer value	xsd:int	int
integer	An arbitrary-precision, signed, integer value	xsd:integer	java.math.BigInteger
long	A signed, 64-bit, integer value	xsd:long	long
string	A string of Unicode characters	xsd:string	java.lang.String

MessageTypeEnum

```
MessageTypeEnum: { INFORMATION, WARNING, ERROR }
```

MessageTypeEnum is an enumerated type that defines all possible message types.

- INFORMATION: an informational message
- WARNING: a warning message
- ERROR: an error message

Message

```
Message: [MessageTypeEnum type, string code, string localizedText, string
 logDetail]
```

Message is a data structure that defines the result of a webservice API call. It provides optional fields for a non-localized code, localized text, and log detail. Currently, all localized text uses the locale that is set for the Unica Plan server instance.

Table 5. Message parameters

Parameter	Description
type	A MessageTypeEnum, setting the type of the message.
code	An optional code, in string format, for the message.
localizedText	An optional text string to associate with the message.
logDetail	An optional stack trace message.

NameValue

```
NameValue: [string name, int sequence]
```

NameValue is a base complex type that defines a name-value pair. It also defines an optional sequence that the service uses to construct value arrays as needed (the sequences are zero-based).

All NameValues with the same name, but different sequence numbers, are converted into an array of values and associated with the common name.

The array size is determined by the maximum sequence number; unspecified array elements have null values. Array sequence numbers must be unique. The value and its type are provided by the extended type.

Table 6. NameValue parameters

Parameter	Description
name	A string that defines the name of a NameValue type.
sequence	A zero-based integer that sets the sequence number for the Name-Value implied value.

Extended NameValue types are defined for each primitive type, as follows:

Table 7. Extended NameValue types

Extended type	Description
BigDecimalNameValue: NameValue [decimal value]	A NameValue type whose value is an arbitrary-precision, decimal number.
BigIntegerNameValue: NameValue [integer value]	A NameValue type whose value is an arbitrarily sized integer.
BooleanNameValue: NameValue [Boolean value]	A NameValue type whose value is a Boolean.
CurrencyNameValue: NameValue [string locale, decimal value]	A NameValue type suitable for representing currency in some locale. Locale is an ISO Language Code, that is, the lowercase, two-letter codes as defined by ISO-639. Currently, the locale must agree with the locale set in the Unica Plan server instance.
DateNameValue: NameValue [datetime value]	A NameValue type whose value is a date.
DecimalNameValue: NameValue [double value]	A NameValue type whose value is a double-precision, decimal number.

Table 7. Extended NameValue types (continued)

Extended type	Description
IntegerNameValue: NameValue [long value]	A NameValue type whose value is a 64-bit integer.
String NameValue: NameValue [string value]	A NameValue type whose value is a string.

And finally, an array of the extended NameValue types is defined for use when you must define a set of NameValues of with different types.

```
    NameValueArrays: [
BooleanNameValue[]     booleanValues,
StringNameValue[]      stringValues,
IntegerNameValue[]     integerValues,
BigIntegerNameValue[] bigIntegooleanNameValue,
DecimalNameValue[]     decimalValues,
BigDecimalNameValue[] bigDecimalValues
DateNameValue[]        dateNameValues
CurrencyNameValue[]   currencyValues
     ]
```

Chapter 3. Unica Plan procedures

A "procedure" is a custom or standard Java™ class hosted by Unica Plan that does some unit of work. Procedures provide a way for customers and Professional Services to extend business logic in arbitrary ways.

Procedures follow a simple programming model with a well-defined API to affect components that are managed by Unica Plan. Procedures are "discovered" through a simple lookup mechanism and XML-based definition file. Unica Plan runs the procedures according to needs of their "clients." For example, in response to an integration request (incoming) or a trigger firing (internal or outgoing).

Procedures run synchronously with their client; results are made available directly to the client, and through a persisted auditing mechanism. The execution of a procedure can also cause other events and triggers to fire in Unica Plan.

Procedures must be written in Java™.

Assumptions

The procedure implementation classes are packaged into a separate classes tree or JAR file and made available to Unica Plan through a URL path.

Procedure implementation

The procedure execution manager uses an independent class loader to load these classes as needed. By default, Unica Plan looks in the following directory.

```
<Plan_Home>/devkits/integration/examples/classes
```

To change this default, set the **integrationProcedureClasspathURL** parameter under **Settings > Configuration > Plan > umoConfiguration > integrationServices**.

The procedure implementation class name follows the accepted Java™ naming conventions, to avoid package collisions with "unica" and classes from other vendors. In particular, customers must not place procedures under the "com.unica" or "com.unicacorp" package tree.

The procedure implementation is coded to the Java™ runtime version used by Unica Plan on the application server (at least JRE 1.8).

The procedure implementation class is loaded by the class loading policy that is normally used by Unica Plan (typically **parent-last**). The application server might provide development tools and options to reload classes that would apply to Unica Plan procedures, but that is not required.

Libraries

Unica Plan provides some open source and third-party libraries; application servers also use different versions of these libraries.

Generally, this list changes from release to release. The following third-party libraries are supported.

- `activation.jar`
- `axiom-api-1.2.15.jar`
- `axiom-compat-1.2.15.jar`
- `axiom-dom-1.2.15.jar`
- `axiom-impl-1.2.15.jar`
- `axis2-adb-1.5.2.jar`
- `axis2-adb-codegen-1.5.2.jar`
- `axis2-codegen-1.5.2.jar`
- `axis2-kernel-1.5.2.jar`
- `axis2-transport-http-1.5.2.jar`
- `axis2-transport-local-1.5.2.jar`
- `httpcore-4.0.jar`
- `commons-codec.jar`
- `commons-httpclient-3.1.jar`
- `commons-lang.jar`
- `commons-logging.jar`
- `disruptor-3.4.2.jar`
- `geronimo-stax-api_1.0_spec-1.0.1.jar`
- `httpclient-4.3.6.jar`

- `httpcore-4.3.3.jar`
- `jersey-client-1.17.jar`
- `jersey-core-1.17.jar`
- `jersey-json-1.17.jar`
- `junit-4.4.jar`
- `log4j.jar`
- `log4j-api-2.8.2.jar`
- `log4j-core-2.8.2.jar`
- `mail.jar`
- `neethi-2.0.4.jar`
- `wsdl4j-1.6.2.jar`
- `xlxpScanner.jar`
- `xlxpScannerUtils.jar`
- `xlxpWASParsers.jar`
- `XmlSchema-1.4.3.jar`
- `Unica Plan APIs latest version (affinium_plan.jar)`
- `Unica Platform APIs latest version (unica-common.jar)`

If a procedure, or the secondary classes the procedure imports, does use such packages, their use must agree exactly with the packages provided by Unica Plan or the application server. In this case, rework of your procedure code is required if a later version of Unica Plan upgrades or abandons a library.

Procedures and threads

The procedure must be thread-safe concerning its own state; that is, its run method cannot depend on internal state changes from call to call. A procedure cannot create threads on its own.

Configuration parameters

When you install the Unica Plan Integration Module, the installer sets three configuration properties. You can modify the configuration properties to customize the behavior of the Integration Module.

Configuration properties for the Integration Module are under **Plan | umoConfiguration | integrationServices**.

- The **enableIntegrationServices** configuration property to turns the Integration Services module on and off.
- The **integrationProcedureDefinitionPath** parameter contains the full file path to the custom procedure definition XML file.

  The default value is `<HCL_Unica_Home><Plan_Home>/devkits/integration/ examples/src/procedure/procedure-plugins.xml/`.

- The **integrationProcedureClasspathURL** parameter contains the URL to the class path for custom procedures.

  The default value is `file:///<HCL_Unica_Home><Plan_Home>/devkits/ integration/examples/classes/`.

  Note: The '/' at the end of the `integrationProcedureClasspathURL` path is required for loading procedure classes correctly.

Design

The procedure implementation class uses the Unica Plan API to read and update Unica Plan components, start services, and so on. Other Java™ packages can be used to do other tasks.

In your design, focus on producing a single unit of work that operates atomically. Ideally, a procedure performs some series of tasks that can be scheduled asynchronously to run at some later time. This "fire and forget" integration model results in the least load on both systems.

Note: Only the documented classes and methods will be supported in future releases of Unica Plan. Consider all other classes and methods in Unica Plan to be off-limits.

After you code and compile the procedure implementation classes, you make them available to Unica Plan. The build scripts that are supplied with the Unica Plan Integration Services place the compiled procedures in the default location. The final development step is to update the custom procedure plug-in definition file that is used by Unica Plan to discover the custom procedures.

The procedure must implement the **com.unica.publicapi.plan.plugin.procedure.IProcedure** interface and have a parameter-less constructor (usual JavaBeans model). Coding and compilation of each procedure is done in a Java™ IDE of the customer's choice, such as Eclipse, Borland JBuilder, or Idea. Sample code is provided with Unica Plan as developer toolkits, in the following location:

```
<Plan_Home>/devkits/integration/examples/src/procedure
```

Procedure lifecycle

Each procedure runs through a complete lifecycle.

The runtime lifecycle of a procedure includes the following steps.

1. Discovery and initialization
2. Selection (optional)
3. Execution
4. Destruction

Discovery and initialization

Unica Plan must be made aware of all standard and custom procedures available for a particular installation instance. This process is called discovery.

Note: Standard procedures (procedures that are defined by the Unica Plan engineering team) are known implicitly and so do not need any action to be discovered.

Custom procedures are defined in the procedure plug-in definition file. The Unica Plan plug-in manager reads this file during initialization. For each procedure found, the plug-in manager completes the following steps.

1. Instantiate the procedure; transition its state to INSTANTIATED.
2. Create a procedure audit record.
3. If the procedure was instantiated, its **initialize()** method is called with any initialization parameters found in its plug-in description file. If this method throws an exception, the status is logged and the procedure is abandoned. Otherwise, the procedure state changes to the INITIALIZED state. It is now ready to run.
4. Create a procedure audit record.
5. If the procedure was initialized, its **getKey()** method is called to determine the key that is used by clients to reference the procedure. This key is associated with the instance and saved for later lookup.

Selection

From time to time, Unica Plan might present a list of available procedures to users, for example to enable administrators to set up a trigger. Unica Plan only presents this list after the procedure is initialized, using the procedure's **getDisplayName()** and **getDescription()** methods.

Execution

At some point after the procedure is initialized, Unica Plan receives a request to run the procedure. This request might happen concurrently with other procedures (or the same procedure) running on other threads.

At run time, the procedure execution manager completes the following steps.

1. Start a database transaction.
2. Set the procedure state to EXECUTING.
3. Create a procedure audit record.
4. Call the procedure's **execute()** method with an execution context and any run parameters that are provided by the client. The method implementation uses the

Unica Plan API as needed, acquiring edit locks, and passing along the execution context. If the run method throws an exception, the execution manager marks the transaction for rollback.

5. Commit or rollback the transaction according to the execution results; set procedure state to EXECUTED.

6. Release any outstanding edit locks.

7. Create a procedure audit record.

> ✏️ **Note:** The **execute()** method is not intended to alter the procedure instance data.

Destruction

When Unica Plan shuts down, the procedure plug-in manager walks through all loaded procedures. For each procedure found, it completes the following steps.

1. Calls the procedure's destroy() method to allow the procedure to clean up before the instance is destroyed.

2. Changes the state of the procedure to FINALIZED (it cannot be run).

3. Creates a procedure audit record.

4. Destroys the instance of the procedure.

Key Java™ classes

The supplied integration developer's kit contains a set of Javadoc for the public Unica Plan API and supporting classes.

The most important Java classes are listed here.

- IProcedure (com.unica.publicapi.plan.plugin.procedure.IProcedure): interface that all procedures must implement. Procedures go through a well-defined lifecycle and access the Unica Plan API to do work.
- ITriggerProcedure (com.unica.publicapi.plan.plugin.procedure.ITriggerProcedure): interface that all trigger procedures must implement (marker interface).

- IExecutionContext (com.unica.publicapi.plan.plugin.procedure.IExecutionContext): interface of opaque context object that is handed to the procedure by the execution manager. This object has public methods for logging and edit lock management. The procedure also passes this object to all PlanAPI calls.
- IPlanAPI (com.unica.publicapi.plan.api.IPlanAPI): interface to the Unica Plan API. The execution context provides a **getPlanAPI()** method to retrieve the proper implementation.

Data locking

Unica Plan uses a pessimistic edit locking scheme; that is, only one user is granted update access to component instances at a time. For the GUI user, this locking is done at the visual tab level. In some cases, data is locked for a subset of an instance, for example, a project summary tab. In other cases, data is locked across many instances, for example, the workflow tab. After a user acquires a lock, all other users are restricted to read-only access to the related data.

To ensure that the changes made by a procedure to a component instance or group of instances are not inadvertently overwritten by another user, a procedure must acquire the appropriate locks before it updates component data. The execution context object that is passed to the procedure's **execute()** method is used to accomplish lock the data.

Before the procedure updates any data, it must call the context's **acquireLock()** method for each lock it needs. For example, if a procedure is going to update a project and the associated workflow, the procedure must acquire locks for both.

If another user already has a lock, the **acquireLock()** method throws a **LockInUseException** immediately. To minimize collisions, the procedure must release the lock as soon as it updates the object.

The execution manager automatically releases any outstanding locks when the execute method returns. In any case, locks are only held for the life of the database transaction. That is, locks expire if the database-specific transaction timeout is exceeded.

Note: Edit locks are not the same as database transactions.

Procedure transactions

The procedure execution manager automatically wraps execution of the procedure with a database transaction, committing or rolling it back based on the outcome of the procedure execution.

Wrapping the procedure execution and database transaction ensures that updates to the Unica Plan database are not visible to other users until committed. It also makes the updates atomic.

The procedure writer still must acquire the necessary edit locks to ensure that other users cannot write changes to the database before the procedure execution completes.

Procedure communication

The **execute()** method of a procedure returns an integer status code to the Unica Plan procedure audit table. The **execute()** method of a procedure can also return zero or more messages to the procedure audit table, which are logged and persisted.

The client might also communicate the status information in some other way.

Procedure logging

Unica Plan has a separate log file for procedures: `<Plan_Home>\logs\system.log`

The procedure execution manager logs the lifecycle of each procedure and creates audit records.

- **logInfo()**: write an informational message to the procedure log.
- **logWarning()**: write a warning message to the procedure log.
- **logError()**: write an error message to the procedure log.
- **logException()**: dump the stack trace for the exception to the procedure log.

Procedure plug-in definition file

The procedure plug-in definition file defines implementation class, metadata, and other information about the custom procedures to be hosted in Unica Plan.

By default, the procedure plug-in definition is assumed to be in the following path:

`<Plan_Home>/devkits/integration/examples/src/procedures/procedure-plugins.xml`

This file is an XML document that contains the following information.

Procedures: a list of zero or more **Procedure** elements.

Procedure: an element that defines a procedure. Each procedure contains the following elements.

- **key** (optional): string that defines the lookup key for the procedure. This key must be unique among all standard and custom procedures that are hosted by a particular Unica Plan instance. If not defined, defaults to the fully qualified version of the **className** element. Names starting with the string "uap" are reserved for use by Unica Plan.
- **className** (required): fully qualified package name of the procedure class. This class must implement the IProcedure class (com.unica.public.plan.plugin.procedure.IProcedure).
- **initParameters** (optional): a list of zero or more initParameter elements.

  initParameter(optional): parameter to be passed to the procedure's initialize() method. This element includes the nested parameter name, type, and value elements.

  - name: string that defines the parameter name
  - type: optional class name of the Java™ wrapper class that defines the type of the parameter value. Must be one of the following types:
    - java.lang.String (the default)
    - java.lang.Integer
    - java.lang.Double
    - java.lang.Calendar
    - java.lang.Boolean
  - value: string form of the attribute value according to its type

Chapter 4. Unica Plan SOAP API

The Unica Plan SOAP API is a façade that provides a client view of a running Unica Plan instance.

Only a subset of the Unica Plan capabilities is shown to users. The API is used concurrently by Unica Plan web users and Unica Plan Integration Services WebService SOAP requests and triggers. The API supports the following types of operations.

- Component creation and deletion
- Discovery (by component type, attribute value, and more values)
- Component inspection (through its attributes, specialized links, and more values)
- Component modification

Note: Unica Plan APIs are intended for Administrator use only.

Contents of the Unica Plan SOAP API

The com.unica.publicapi.plan.api package delivers the Unica Plan SOAP API.

This package offers interfaces and exceptions, and contains the following types of classes:

- Enumerated data types.
- Handles to identify object and component instances.
- A Java™ map, AttributeMap.

Complete documentation of the API, including all methods and possible values, is available by clicking **Help > Product Documentation** in an instance of Unica Plan, then downloading the HCL<version>PublicAPI.zip file.

SOAP API interfaces

The Unica Plan SOAP application programming interface (API) includes **IPlanAPI** and **IExecutionContext**.

The Unica Plan SOAP API includes the following interfaces.

IPlanAPI

Defines the public API for Unica Plan. Provides methods for creating, discovering, and modifying objects, including folders, projects, programs, workflow tasks, and team members.

For systems that have the optional integration with Unica Campaign enabled, also provides methods for creating, discovering, and modifying offers.

IExecutionContext

Defines the triggers and locks that execute methods in the API.

API methods

For specific information about the public API methods, refer to the iPlanAPI class in the JavaDocs API documentation files.

These files are available by logging in to Unica Plan and selecting **Help > Product Documentation** from any page, and then downloading the `<version>PublicAPI.zip` file.

SOAP API common exceptions

Common exceptions that are thrown by the SOAP API include NotFoundException, AuthorizationException, DataException, InvalidExecutionContextException, and NotLockedException.

The following list explains why these common exceptions might occur.

- *<object type>*NotFoundException: The system is unable to return the specified item or object.
- AuthorizationException: The user who is associated with the execution context is not authorized for the requested operation. This exception can be thrown by any API method, so is undeclared.
- DataException: An exception occurred in the underlying database layer in Unica Plan. Check the SQL log for details.

- InvalidExecutionContextException: There is a problem with an execution context passed to an API method (for example, the method was not initialized correctly). This exception can be thrown by any API, so is undeclared.
- NotLockedException: Attempt to update component data without first acquiring the required lock. See the acquireLock() method of the IExecutionContext interface.

SOAP API handles

A handle is special URL object that references a particular object instance in an Unica Plan instance. Handles include the component type, internal data identifier, and an instance base URL.

Handles used or generated by the API can be externalized to a full URL. You can use the resulting URL in different ways. You can use the URL to open a view of the component in the Unica Plan GUI, send it in email messages, or use it in another procedure as a parameter.

Handles are valid only for a particular Unica Plan service instance or clustered instance, but are valid for the lifetime of the deployed service. As a result, handles can be saved in a file for later reference, but they cannot be used to access components on another Unica Plan instance. This restriction also applies to instances on the same physical host server. Unica Plan does provide, however, a mechanism for mapping different base URLs to the current instance to accommodate relocating an instance to another server (for example, if the equipment malfunctions).

Handles are client-independent. For example, a trigger can pass a handle to a procedure, which uses it as a parameter in a SOAP call to a 3rd-party system. The 3rd-party system can then issue a SOAP request back to Unica Plan to start a procedure that updates an attribute.

Members of the Handle class have factory methods for creating handles from various types of URLs. Examples follow.

Approval

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=approvaldetail&
approvalid=101
```

Asset

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=asset&
assetMode=VIEW_ASSET&assetid=101
```

Asset Folder

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=folder&id=101
```

Asset Library

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=library&id=101
```

Attachment

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=attachmentview&
attachid=101&parentObjectId=101&parentObjectType=project
```

Financial Account

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=accountdetails&
accountid=101
```

Folder

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=grouping_folder&
folderid=1234
```

Invoice

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=invoicedetails&
invoiceid=134
```

Invoice line item

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=invoicedetails&
invoiceid=134&line_item_id=101
```

Marketing object

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=componenttabs&
componentid=creatives&componentinstid=1234
```

Marketing object grid

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=componenttabs&
componentid=creatives&componentinstid=1234&gridid=grid
```

Marketing object grid row

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=componenttabs&
componentid=creatives&componentinstid=1234&gridid=grid&gridrowid
=101
```

Plan team

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=teamdetails&
func=edit&teamid=100001
```

Plan user

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=adminuserpermiss
ions&
func=edit&userId=101
```

Program

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=programtabs&prog
ramid=125
```

Program grid

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=programtabs&
programid=1234&gridid=grid
```

Program grid row

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=programtabs&
programid=1234&gridid=grid&gridrowid=101
```

Project

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=projecttabs&
projectid=1234
```

Project grid

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=projecttabs&
projectid=1234&gridid=grid
```

Project grid row

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=projecttabs&
projectid=1234&gridid=grid&gridrowid=101
```

Project line item

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=projecttabs&
projectid=1234&projectlineitemid=123&projectlineitemisversionfin
al=false
```

Workflow stage

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=projectworkflow&
projectid=1234&taskid=5678
```

Workflow task

```
http://mymachine:7001/plan/affiniumplan.jsp?cat=projectworkflow&
projectid=1234&taskid=5678
```

SOAP API AttributeMap

The AttributeMap class is a Java™ map that contains only attributes. The attribute <Name> is the map entry key, and the attribute <values> array (note plural) is the map entry value.

The AttributeMap class includes the following fields.

- *<Name>*: the programmatic name of the attribute. This name serves as a unique key for accessing the attribute within the component instance in which it occurs.

  > **Note:** *<Name>* is not necessarily the display name that is presented to a user in the GUI. For components that are created from templates (such as projects or workflow tasks), the attribute name is specified by the template element definition. The attribute name must be unique. For other components, the attribute name typically is derived programmatically from the server-side component instance (for example, through Java™ introspection).

  > **Note:** By convention, custom attributes include the name of the form in which the editable version is defined: *<form_name>.<attribute_name>*.

- *Values*: a Java™ object array, containing zero or more attribute values. The type of each value must be the same and agree with the type of the attribute as it is defined in Unica Plan. Only the following Java™ wrapper and Unica Plan types are supported:
    - AssetLibraryStateEnum: a AssetLibraryStateEnum enumerated type value.
    - AssetStateEnum: a AssetStateEnum enumerated type value.
    - AttachmentTypeEnum: a AttachmentTypeEnum enumerated type value.
    - AttributeMap: a map that holds attributes.
    - BudgetPeriodEnum: a BudgetPeriodEnum enumerated type value.
    - BudgetTypeEnum: a BudgetTypeEnum enumerated type value.
    - Handle: a reference to a component instance, grid row, attribute, and so on.
    - InvoiceStateEnum: an InvoiceStateEnum enumerated type value.
    - java.io.File: representation of a file.
    - java.lang.Boolean: a Boolean value, either True or False
    - java.lang.Double: a double-precision decimal number value.
    - java.lang.Float: a single-precision decimal number value
    - java.lang.Integer: a 32-bit integer value
    - java.lang.Long: a 64-bit integer value
    - java.lang.Object: Generic Java™ object

- java.lang.String: a string of zero or more Unicode characters
- java.math.BigDecimal: arbitrary-precision signed decimal number value. Suitable for currency; the interpretation of the value depends on the currency locale for the client.
- java.math.BigInteger: arbitrary-precision integer value.
- java.net.URL: a Universal Resource Locator (URL) object.
- java.util.ArrayList: List of objects.
- java.util.Calendar: a date-time value for a particular locale.
- java.util.Date: a date-time value. This type is deprecated. Use java.util.Calendar or java.util.GregorianCalendar instead.

> **Note:** To implement date, users can select either java.util.Calendar or java.util.GregorianCalendar.

- java.util.GregorianCalendar: GregorianCalendar is a concrete subclass of java.util.Calendar and provides the standard calendar system in use by most of the world.
- MonthEnum: a MonthEnum enumerated type value.
- ProjectStateEnum: a ProjectStateEnum enumerated type value.
- QuarterEnum: a QuarterEnum enumerated type value.
- TaskStateEnum: a TaskStateEnum enumerated type value.
- WeekEnum: a WeekEnum enumerated type value.

The metadata of an attribute (such as translated display name and description) is defined by the template that is associated with the attribute and its parent object instance. Attributes provide a simple yet extensible mechanism for showing both required and optional object instance attributes, such as project name, code, and start date.

SOAP API enumerated data types

The Unica Plan SOPA API supports the following enumerated data types and values.

ApprovalMethodEnum

ApprovalMethodEnum defines valid approval methods. Possible values are:

- SEQUENTIAL
- SIMULTANEOUS

ApprovalStateEnum

ApprovalStateEnum defines valid approval states. Possible values are:

- CANCELLED
- COMPLETED
- IN_PROGRESS
- NOT_STATED
- ON_HOLD

AssetLibraryStateEnum

AssetLibraryStateEnum defines valid asset library states. Possible values are:

- DISABLED
- ENABLED

AssetStateEnum

AssetStateEnum defines valid asset states. Possible values are:

- ARCHIVE
- DRAFT
- FINALIZE
- LOCK

AttachmentTypeEnum

AttachmentTypeEnum defines valid attachment types. Possible values are:

- ASSET
- FILE
- URL

BudgetPeriodEnum

BudgetPeriodEnum defines the possible budget periods. Possible values are:

- ALL
- MONTHLY
- QUARTERLY
- WEEKLY
- YEARLY

BudgetTypeEnum

BudgetTypeEnum defines valid budget types. Possible values are:

- ACTUAL
- ALLOCATED
- COMMITTED
- FORECAST
- TOTAL

ComponentTypeEnum

ComponentTypeEnum identifies the accessible Unica Plan component types. Possible values are:

- APPROVAL
- ASSET
- ASSET_FOLDER
- ASSET_LIBRARY
- ATTACHMENT
- FINANCIAL_ACCOUNT
- GROUPING_FOLDER
- INVOICE
- MARKETING_OBJECT
- PLAN_TEAM
- PLAN_USER

- PROGRAM
- PROJECT
- PROJECT_REQUEST
- TASK
-

InvoiceStateEnum

InvoiceStateEnum defines valid invoice states. Possible values are:

- CANCELLED
- DRAFT
- PAID
- PAYABLE

MonthEnum

MonthEnum defines valid values for the month.

OfferStateEnum

OfferStateEnum defines valid offer states. Possible values are:

- STATE_OFFER_DRAFT
- STATE_OFFER_PUBLISHED
- STATE_OFFER_RETIRED

ProjectCopyTypeEnum

ProjectCopyTypeEnum defines valid methods for copying a project. Possible values are:

- COPY_USING_PROJECT_METRICS
- COPY_USING_TEMMPLATE_METRICS

ProjectParticipantLevelEnum

ProjectParticipantLevelEnum identifies the roles that users can have in a project. Possible values are:

- OWNER
- PARTICIPANT
- REQUESTER

ProjectStateEnum

ProjectStateEnum defines valid project and request states. Possible values are:

- ACCEPTED
- CANCELLED
- COMPLETED
- DRAFT
- IN_PROGRESS
- IN_RECONCILIATION
- LATE: the project did not start by its scheduled begin date.
- NOT_STARTED
- ON_HOLD
- OVERDUE: the project was not completed before its scheduled end date.
- RETURNED
- SUBMITTED

For more information about project and task statuses, see the Unica Plan User's Guide.

QuarterEnum

QuarterEnum defines the valid values for quarters: Q1, Q2, Q3, and Q4.

TaskStateEnum

TaskStateEnum defines valid workflow task states. Possible values are:

- ACTIVE
- DISABLED
- FINISHED
- PENDING
- SKIPPED

WeekEnum

WeekEnum defines valid values for weeks in a year, from WEEK_1 to WEEK_53.